Practicum Recursieve Datastructuren

– voorjaar 2006 –

Opgave 8: *SKW introduceert ook een hogesnelheidslijn*

1 Achtergrond

Binnenkort komt SKW (Snel een Kaartje en Wegwezen), een concurrent van de Nederlandse Spoorwegen, met een eigen hogesnelheidlijn. Om voldoende te kunnen concurreren met de NS is het noodzakelijk om de kosten zo laag mogelijk te houden. SKW wil dit doen door de trein niet op alle stations waar reizigers naar toe willen te laten stoppen. Reizigers kunnen dan verder met de stoptrein van de NS. Wat is nu het optimale treinschema met slechts een beperkt aantal stops waarbij reizigers zo min mogelijk met de stoptrein hoeven te reizen om hun bestemming te bereiken?

Dit is typisch een optimalisatie (dan wel minimalisatie) probleem. We kunnen proberen dit op te lossen door, al dan niet met *backtracking*, alle mogelijke treinschema's te bekijken. Dit kan echter erg lang duren en SKW wil pas vlak voor vertrek het schema berekenen zodat kaartjes verkocht kunnen worden tot op het laatste moment. Zoals jullie nu bekent is, kunnen we z'n algoritme versnellen (ten koste van meer geheugenruimte) door tussenresultaten te hergebruiken met behulp van *dynamisch programmeren*.

2 Leerdoelen

Na afloop van deze opdracht ben je in staat om:

- optimalisatie problemen te herkennen als dynamisch programmeren problemen.
- problemen op te lossen met behulp van dynamisch programmeren.

3 Instructie

Bestudeer allereerst hoofdstuk 24 van het dictaat en de sheets over dynamisch programmeren zoals die gebruikt zijn op het college. Begin pas daarna aan de volgende deelopdrachten. Zorg ervoor, dat je (op papier en aan de hand van een of meer plaatjes) al een ontwerp van je datastructuur en optimalisatiealgoritme hebt gemaakt, vóórdat je aanschuift achter de pc om je code in te voeren en uit te testen.

4 Probleemschets 1

Om hogesnelheidstreinen zo efficiënt mogelijk te laten rijden moet er zo min mogelijk gestopt worden om het afremmen en optrekken zo veel mogelijk te beperken. Dit is ook nodig om de trein z'n maximum snelheid te laten halen, want dit kan alleen op langere trajecten. Daarom wil SKW de trein slechts een beperkt aantal keren laten stoppen op het traject. Dit aantal wordt gekozen door SKW en kan van dag tot dag verschillen afhankelijk van de hoeveelheid vocht en bladeren op de rails. Reizigers kunnen daarna hun reis vervolgen met stoptreinen van de NS, waarvoor SKW de NS natuurlijk moet betalen. Aangezien SKW een grootafnemer van kaartjes is wordt er per stoptreinhalte betaald ongeacht de afstand. Soms kost het minder stoptreinhaltes om de reizigers voorbij hun bestemming te vervoeren en dan terug te laten reizen met de stoptrein. Als verder reizen met de stoptrein even duur is als terug reizen, gaan we er vanuit dat de reiziger vóór hun bestemming uitstappen. Om de kosten nog verder te drukken is er geen conducteur of machinist

op de trein aanwezig (de techniek staat voor niets!). Daarom kan men alleen een kaartje kopen en instappen bij het vertrekstation en kan men op alle andere stations alleen uitstappen. Alle informatie over de reizigers is hierdoor voor vertrek beschikbaar en een optimale treinschema kan dus van te voren berekend worden.

We geven alvast de klasse structuur van het programma, dat jullie moeten afmaken, voor:

```
const int Zitplaatsen = 1000;

class Treinschema
{
private:
    int reizigers, stations;
    int bestemming[Zitplaatsen];

    int stoptreinVanaf(int laatste);
    int stoptreinTussen(int vorige, int volgende);

public:
    Treinschema(int _reizigers, int _bestemming[]);
    void goedkoopsteSchema(int stops);
};
```

In de klasse slaan we het aantal reizigers en het nummer van het laatste station op. Voor het gemak worden stations niet aangeduid met hun naam, maar met hun volgnummer vanaf het vertrekstation (station 0).

Deelopdracht 1

Implementeer de constructor die het aantal `reizigers` en hun `bestemmingen` opslaat in velden van de klasse, zodat de andere methoden deze kunnen gebruiken. `bestemmingen` is een rij die voor iedere reiziger het nummer van het gewenste station bevat. Het laatste station waar reizigers naar toe willen, ook handig om te weten en op te slaan in `stations`, kun je afleiden uit deze rij.

Deelopdracht 2

Voor het berekenen van de kosten die SKW aan de NS moet betalen (en die we willen minimaliseren) gebruiken we twee hulpfuncties. Implementeer de functie `stoptreinVanaf` die bepaalt hoeveel SKW de NS moet betalen voor alle reizigers die vanaf het `laatste` station (de hogesnelheidstrein rijdt niet verder) nog verder moeten met de stoptrein. Maak ook de functie `stoptreinTussen` die de minimale kosten aan NS–kaartjes bepaalt die alle reizigers met bestemmingen tussen de `vorige` stop (exclusief) en de `volgende` (inclusief) stop samen nog nodig hebben.

5 Probleemschets 2

Het aantal stops is bepaald door SKW en ligt dus vast. Zoals bij alle dynamisch programmeren oplossingen gebruik je een tabel die je iteratief vult. De tabel kun je initialiseren door voor 0 stops de functie `stoptreinVanaf` te gebruiken om uit te rekenen hoeveel het kost als de hogesnelheidstrein helemaal niet rijdt, en men de stoptrein vanaf de vertrekstation moet nemen. De oplossingen voor N stops kun je berekenen aan de hand van de oplossingen voor $N-1$ stops. De kosten voor de N^{de} stop bij station $S_{volgende}$ hangt af van de kosten voor de $N-1^{ste}$ stop bij het vorige station S_{vorige}. Het verschil bestaat uit: 1) hoeveel je bespaart doordat reizigers nu niet vanaf station S_{vorige} met de stoptrein naar hun bestemming hoeven, 2) hoeveel het kost om reizigers met een bestemming tussen station S_{vorige} en station $S_{volgende}$ met de stoptrein te laten

reizen, 3) de kosten voor de reizigers die vanaf station $S_{volgende}$ nog naar hun bestemming moeten. Als de tabel gevuld is kun je de goedkoopste oplossing bij N stops eenvoudig opzoeken.

Deelopdracht 3

Implementeer de procedure `goedkoopsteSchema` die de minimale kosten afdrukt bij een bepaald aantal `stops`.

Probeer je programma uit met bijvoorbeeld met 10 reizigers uniform verdeelt over 11 stations (maar natuurlijk niet het vertekstation 0): `bestemming[i] == i+1`. Één stop zou station 7 met een prijs van 18 stoptreinhaltes moeten opleveren. Twee stops stations 3 en 8 met 11 stoptreinhaltes, en drie stops stations 3, 6 en 9 met 7 stoptreinhaltes.

Deelopdracht 4

Pas je implementatie aan zodat 'ie ook de stations waar gestopt wordt (in het goedkoopste traject) afdrukt. Bijvoorbeeld door ook een tabel bij te houden die per stop de index van de vorige stop geeft die bij een minimale oplossing hoort.

6 Producten

Als producten moet je schetsen hebben van de gebruikte datastructuren en (uitgeteste!) C++-code, die voldoet aan de gevraagde specificaties en aan de kwaliteitscriteria zoals die gesteld worden (zie bij punt 7. Zelfreflectie). Lever ook de uitvoer van je programma in (als commentaarregels, achter je broncode in het `.cpp`-bestand). De schetsen van de datastructuren dien je ter goedkeuring (tijdens het practicum) aan de studentassistenten voor te leggen.

7 Zelfreflectie

Via de cursuswebpagina kom je uit op een aantal criteria die we ook bij deze cursus gebruiken voor de kwaliteit van je ingeleverde werk. Ga voor jezelf na, voor zover dat nu al van toepassing is, in hoeverre je uitwerkingen voldoen aan deze criteria. Je uitwerking van deze zelfreflectie moet je echter niet inleveren.

8 Inleveren van je producten

Vóór dinsdag 9 mei, 9.00 uur, en wel door jullie programma één keer via e-mail met de tekst `P2 opgave8` in de `Subject:`-regel naar **`S.Smetsers@science.ru.nl`** te sturen. Vergeet niet om **in** de uitwerking duidelijk jullie namen te vermelden.